

# Learning to Paint With Model-based Deep Reinforcement Learning

Zhewei Huang<sup>1,2</sup> Wen Heng<sup>1</sup> Shuchang Zhou<sup>1</sup>  
<sup>1</sup>Megvii Inc <sup>2</sup>Peking University  
 {huangzhewei, hengwen, zsc}@megvii.com

## Abstract

We show how to teach machines to paint like human painters, who can use a small number of strokes to create fantastic paintings. By employing a neural renderer in model-based Deep Reinforcement Learning (DRL), our agents learn to determine the position and color of each stroke and make long-term plans to decompose texture-rich images into strokes. Experiments demonstrate that excellent visual effects can be achieved using hundreds of strokes. The training process does not require the experience of human painters or stroke tracking data. The code is available at <https://github.com/hzwer/ICCV2019-LearningToPaint>.

## 1. Introduction

Painting, being an important form of visual art, symbolizes human wisdom and creativity. In recent centuries, artists have used a diverse array of tools to create their masterpieces. But it’s hard for people to master this skill without spending a large amount of time in proper training. Therefore, teaching machines to paint is a challenging task and helps to shed light on the mystery of painting. Furthermore, the study of this topic can help us build painting assistant tools.

We train an artificial intelligence painting agent that can paint strokes on a canvas in sequence to generate a painting that resembles a given image. Neural networks are used to produce parameters that control the position, shape, color, and transparency of strokes. Previous works have studied teaching machines to learn painting-related skills, such as sketching [7, 3, 29], doodling [35] and writing characters [34]. In contrast, we aim to teach machines to handle more complex tasks, such as painting portraits of humans and natural scenes in the real world, which have rich textures and complex structural compositions.

We address three challenges for training an agent to paint real-world images. First, painting like humans requires an agent to have the ability to decompose a given target image into an ordered sequence of strokes. The agent needs to

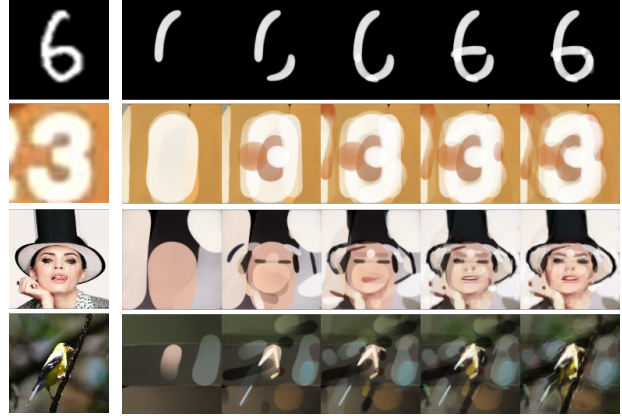


Figure 1: **The painting process.** The first column shows the target images. Our agent tends to draw in a coarse-to-fine manner.

parse the target image visually, understand the current status of the canvas, and have foresightful plans about future strokes. To achieve this planning, one method is to give the supervised loss for stroke decomposition at each step, as in [7]. However, such a method requires ground truth stroke decomposition, which is hard to define. Also, texture-rich image painting usually requires hundreds of strokes to generate a painting that resembles a target image, which is tens of times more than doodling, sketching or character writing and increases the difficulty of planning. To handle the ill-definedness of the problem, and the long-term planning challenge, we propose using reinforcement learning (RL) to train the agent, because RL can maximize the cumulative rewards of a whole painting process rather than minimizing supervised loss at each step. Experiments show that an RL agent can build plans for stroke decomposition with hundreds of steps. Moreover, we apply the adversarial training strategy [5] to improve the pixel-level quality of the generated images, as the strategy has proved effective in other image generation tasks [17].

Second, we design continuous stroke parameter space, including stroke location, color, and transparency, to improve the painting quality. Previous works [7, 35, 4] de-

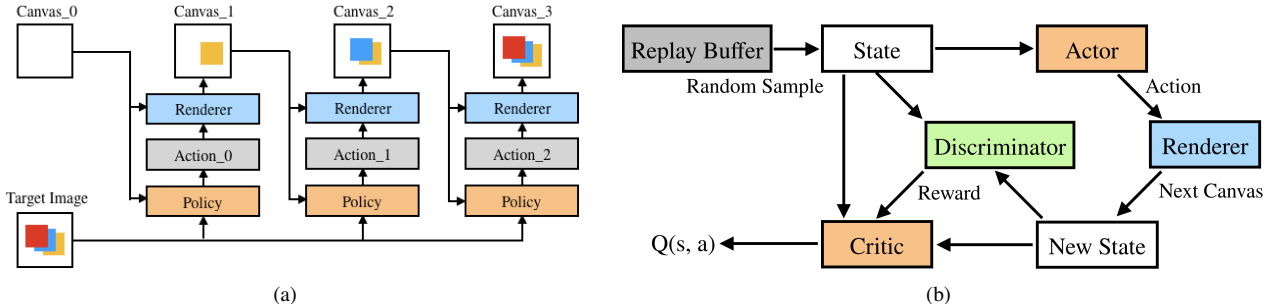


Figure 2: **The overall architecture.** (a) At the inference stage, the actor outputs a set of stroke parameters based on the canvas status and target image at each step. The renderer then renders the stroke on the canvas accordingly. (b) At the training stage, the actor is trained with assistants of an adversarial discriminator and a critic. The reward is given by the discriminator at each step, and the training samples are randomly sampled from the replay buffer.

sign discrete stroke parameter spaces and each parameter has only a limited number of choices, which fall short for texture-rich paintings. Instead, we adopt the Deep Deterministic Policy Gradient (DDPG) [19] which copes well with the continuous action space of the agent.

Third, we build an efficient differentiable neural renderer that can simulate painting of hundreds of strokes on the canvas. Most previous works [7, 35, 4] paint by interacting with undifferentiable painting simulation environments, which are good as renders but fail to provide detailed feedback about the generated images. Instead, we train a neural network that directly maps stroke parameters to stroke paintings. The renderer can also be adapted to different stroke designs like triangle and circles by changing the generation patterns. Moreover, the differential renderer can be combined with DDPG into a single model-based DRL that can be trained in an end-to-end fashion, which significantly boosts both the painting quality and convergence speed.

In summary, our contributions are three-fold:

- We approach the painting task with the model-based DRL algorithm and build agents that decompose the target image into hundreds of strokes in sequence which can recreate a painting on canvas.
- We build differentiable neural renderers for efficient painting and flexible support of different stroke designs, *e.g.* Bézier curve, triangle, and circle. The neural renderer contributes to the painting quality by allowing training model-based DRL agent in an end-to-end fashion.
- Experiments show that the proposed painting agent can handle multiple types of target images well, including handwritten digits, streetview house numbers, human portraits, and natural scene images.

## 2. Related work

Stroke-based rendering (SBR) is a method of non-photorealistic imagery that recreates images by placing discrete drawing elements such as paint strokes or stipples [12] on canvas. Most SBR algorithms solve the stroke decomposition problem by Greedy Search on every single step or require user interaction. Haerberli *et al.* [9] propose a semi-automatic method which requires the user to set parameters to control the shape of the strokes and select the positions for each stroke. Litwinowicz *et al.* [21] propose a single-layer painter-like rendering which places the brush strokes on a grid in the image plane, with randomly perturbed positions. Some work also studies the effects of using different stroke designs [11] and the related problem of generating animations from video [20].

Recent works use RL to improve the stroke decomposition of images. SPIRAL [4] is an adversarially trained DRL agent that learn structures in images, but fails to recover the details of human portraits. StrokeNet [34] combines differentiable renderer and recurrent neural network (RNN) to train agents to paint but fails to generalize on color images. Doodle-SDQ [35] trains the agents to emulate human doodling with DQN. Earlier, Sketch-RNN [7] uses sequential datasets to achieve good results in sketch drawings. Artist Agent [32] explores using RL for the automatic generation of a single brush stroke.

## 3. Painting Agent

### 3.1. Overview

The goal of the painting agent is decomposing the given target image into strokes that can recreate the image on the canvas. To imitate the painting process of humans, the agent is designed to predict the next stroke based on observing the current state of the canvas and the target image. However, the stroke at each step needs to be well compatible



Figure 3: The painting results on multiple datasets. The stroke numbers of the paintings are 5, 40, 200 and 400 for MNIST, SVHN, CelebA and ImageNet respectively.

with previous strokes and future strokes to reduce the number of strokes for finishing the painting. We postulate that the agent should maximize the cumulative rewards after finishing the given number of strokes, rather than the gain of current stroke. To achieve this delayed-reward design, we employ a DRL framework, with the diagrams for the overall architecture shown in Figure 2.

In the framework, we model the painting process as a sequential decision-making task, which is described in Section 3.2. And to build the feedback mechanism, we use a neural renderer to help generate detailed rewards for training the agent, which is described in Section 3.3.

### 3.2. The Model

Given a target image  $I$  and an empty canvas  $C_0$ , the agent aims to find a stroke sequence  $(a_0, a_1, \dots, a_{n-1})$ , where rendering  $a_t$  on  $C_t$  can get  $C_{t+1}$ . After rendering these strokes in sequence, we get the final painting  $C_n$ , which should be visually similar to  $I$  as much as possible. We model this task as a Markov Decision Process with a state space  $\mathcal{S}$ , an action space  $\mathcal{A}$ , a transition function  $\text{trans}(s_t, a_t)$  and a reward function  $r(s_t, a_t)$ . The details of these components are specified next.

**State and Transition Function** The state space is constructed by all possible information that the agent can observe in the environment. We separate a state into three parts: states of the canvas, the target image, and the step number. Formally,  $s_t = (C_t, I, t)$ .  $C_t$  and  $I$  are bitmaps and the step number  $t$  acts as additional information to in-

struct the agent the remaining number of steps. The transition function,  $s_{t+1} = \text{trans}(s_t, a_t)$  gives the transition process between states, which is implemented by painting a stroke on the current canvas.

**Action** An action  $a_t$  of the painting agent is a set of parameters that control the position, shape, color and transparency of the stroke that would be painted at step  $t$ . We define the behavior of an agent as a policy function  $\pi$  that maps states to deterministic actions, i.e.  $\pi: \mathcal{S} \rightarrow \mathcal{A}$ . At step  $t$ , the agent observes state  $s_t$  before predicting the parameters of the next stroke  $a_t$ . The state evolves based on the transition function  $s_{t+1} = \text{trans}(s_t, a_t)$ , which runs for  $n$  steps.

**Reward** Selecting a suitable metric to measure the difference between the current canvas and the target image is found to be crucial for training a painting agent. The reward is designed as follows,

$$r(s_t, a_t) = L_t - L_{t+1} \quad (1)$$

where  $r(s_t, a_t)$  is the reward at step  $t$ ,  $L_t$  is the measured loss between  $I$  and the  $C_t$  and  $L_{t+1}$  is the measured loss between  $I$  and the  $C_{t+1}$ . In this work,  $L$  is formulated as the discriminator score that is defined in Section 3.3.3.

To make the final canvas resemble the target image, the agent should be driven to maximize the cumulative rewards in the whole episode. At each step, the objective of the agent is to maximize the sum of discounted future reward  $R_t = \sum_{i=t}^T \gamma^{(i-t)} r(s_i, a_i)$  with a discounting factor  $\gamma \in [0, 1]$ .

### 3.3. Learning

In this section, we introduce how to train the agent using the model-based DDPG algorithm.

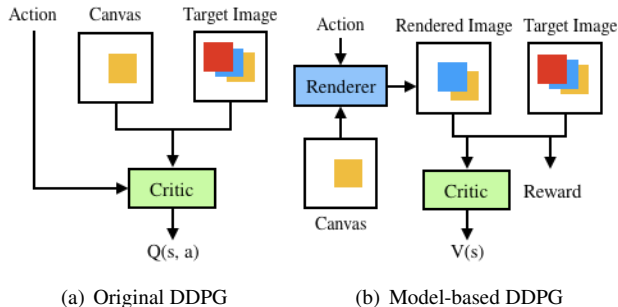


Figure 4: In the original DDPG, critic needs to learn to model the environment implicitly. In the model-based DDPG, the environment is explicitly modeled through a neural renderer, which helps to train an agent efficiently.

#### 3.3.1 Model-based DDPG

We first describe the original DDPG, then introduce building model-based DDPG for efficient agent training.

As we use continuous parameters for strokes, the action space in the painting task is continuous and of high dimensions. Discretizing the action space to adapt some DRL methods, such as DQN and PG, will lose the precision of stroke representation and require many efforts in manual structure design to cope with the explosion of parameter combinations in discrete space. In contrast, DPG [28] uses deterministic policy to resolve the difficulties caused by high-dimensional continuous action space, and DDPG is its variant using Neural Networks.

In the original DDPG, there are two networks: the actor  $\pi(s)$  and critic  $Q(s, a)$ . The actor models a policy  $\pi$  that maps a state  $s_t$  to action  $a_t$ . The critic estimates the expected reward for the agent taking action  $a_t$  at state  $s_t$ , which is trained using Bellman equation (2) as in Q-learning [30] and the data is sampled from an experience replay buffer:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi(s_{t+1})) \quad (2)$$

Here  $r(s_t, a_t)$  is a reward given by the environment when performing action  $a_t$  at state  $s_t$ . The actor  $\pi(s_t)$  is trained to maximize the critic’s estimated  $Q(s_t, \pi(s_t))$ . In other words, the actor decides a stroke for each state. Based on the current canvas and the target image, the critic predicts an expected reward for the stroke. The critic is optimized to estimate more accurate expected rewards.

We cannot train a good-performance painting agent using original DDPG because it’s hard for the agent to model

the complex environment well that is composed of any types of real-world images during learning. The World Model [8] is a method to make agent understand the environments effectively. Similarly, we design a neural renderer so that the agent can observe a modeled environment. Then it can explore the environment and improve its policy efficiently. We term the DDPG with the actor that can get access to the gradients from environments as model-based DDPG. The difference between the two algorithms is visually shown in Figure 4.

The optimization of the agent using the model-based DDPG is different from that using the original DDPG. At step  $t$ , the critic takes  $s_{t+1}$  as input rather than both of  $s_t$  and  $a_t$ . The critic still predicts the expected reward for the state but no longer includes the reward caused by the current action. The new expected reward is a value function  $V(s_t)$  trained using discounted reward:

$$V(s_t) = r(s_t, a_t) + \gamma V(s_{t+1}) \quad (3)$$

Here  $r(s_t, a_t)$  is the reward when performing action  $a_t$  based on  $s_t$ . The actor  $\pi(s_t)$  is trained to maximize  $r(s_t, \pi(s_t)) + V(\text{trans}(s_t, \pi(s_t)))$ . The transition function  $s_{t+1} = \text{trans}(s_t, a_t)$  is the differentiable renderer.

#### 3.3.2 Action Bundle

Frame Skip [2] is a powerful trick for many RL tasks, by restricting the agent to only observe the environment and acts once every  $k$  frames rather than one frame. The trick makes the agents have a better ability to learn associations between more temporally distant states and actions. The agent predicts one action and reuse it at the next  $k - 1$  frames instead and achieves better performance with less computation cost.

Inspired by this trick, we propose using Action Bundle that the agent predicts  $k$  strokes at each step and the renderer renders these strokes in order. This practice encourages the exploration of the action space and action combinations. The renderer can render  $k$  strokes simultaneously to greatly speed up the painting process.

We experimentally find that setting  $k = 5$  is a good choice that significantly improves the performance and the learning speed. It’s worth noting that we modify the reward discount factor from  $\gamma$  to  $\gamma^k$  to keep consistency.

#### 3.3.3 WGAN Reward

GAN has been widely used as a particular loss function in transfer learning, text model and image restoration [18, 33], because of its great ability in measuring the distribution distance between the generated data and the target data. Wasserstein GAN (WGAN) [1] is an improved version of the original GAN that uses the *Wasserstein-1* distance, also

known as *Earth-Mover* distance. The objective of the discriminator in WGAN is defined as

$$\max_D \mathbb{E}_{y \sim \mu} [D(y)] - \mathbb{E}_{x \sim \nu} [D(x)] \quad (4)$$

where  $D$  denotes the discriminator,  $\nu$  and  $\mu$  are the fake samples and real samples distribution. The conditional GAN training schema [14] is used, where fake samples are pairs of a painting and its target; and real samples are two same target images as shown in Figure 5. The prerequisite of the above objective is that  $D$  should be under the constraints of 1-Lipschitz. To achieve the constraint, we use WGAN with gradient penalty (WGAN-GP) [6].

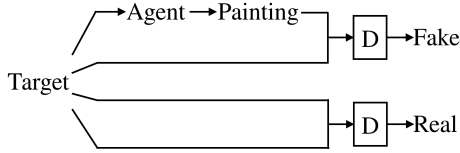


Figure 5: Training of the Discriminator

We want to reduce the differences between paintings and target images as much as possible. To achieve this, we set the difference of  $D$  scores from  $s_t$  to  $s_{t+1}$  using equation (1) as the reward for guiding the learning of the actor. In experiments, we find rewards derived from  $D$  scores is better than  $L_2$  distance.

### 3.4. Network Architectures

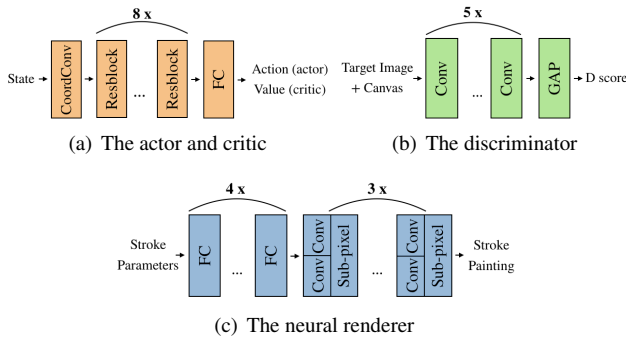


Figure 6: **Network architectures.** FC refers to a fully-connected layer, Conv refers to a convolution layer, and GAP refers to a global average pooling layer. The actor and the critic use the same structure except for the last FC layers that have different output dimensions.

Due to the high variability and complexity of real-world images, we use residual structures similar to ResNet-18 [10] as the feature extractor in the actor and the critic. The actor works well with Batch Normalization (BN) [13], but BN can not speed up the critic learning significantly. We use WN [26] with Translated ReLU (TReLU) [31] on the

critic to stabilize our learning. In addition, we use CoordConv [22] as the first layer in the actor and the critic. For the discriminator, we use a network architecture similar to PatchGAN [14], and with WN and TReLU. The network architectures of the actor, critic and discriminator are shown in Figure 6 (a) and (b).

Following the original DDPG paper, we use the soft target network which creates a copy for the actor and critic and updating their parameters by having them slowly track the learned networks. We also apply this trick on the discriminator to improve its training stability.

## 4. Stroked-based Renderer

In this section, we introduce how to build a neural stroke renderer and use it to generate multiple types of strokes.

### 4.1. Neural Renderer

Using a neural network to generate strokes has two advantages. First, the neural renderer is flexible to generate any styles of strokes and is more efficient on GPU’s than most hand-crafted stroke simulators. Second, the neural renderer is differentiable and enables end-to-end training which boosts the performance of the agent.

Specifically, the neural renderer has as input a set of stroke parameters  $a_t$  and outputs the rendered stroke image  $S$ . The training samples are generated randomly using Computer Graphics rendering programs. The neural renderer can be quickly trained with supervised learning and runs on the GPU. The model-based transition dynamics  $s_{t+1} = \text{trans}(s_t, a_t)$  and the reward function  $r(s_t, a_t)$  are differentiable. Some simple geometric trajectories like circles have simple closed-form gradients. However, in general, the discreteness of pixel position and pixel values requires continuous approximation when deriving gradients, e.g. for Bézier Curves. The approximations need to be carefully designed to not break the agent learning.

The neural renderer is a neural network consisting of several fully connected layers and convolution layers. Sub-pixel upsampling [27] is used to increase the resolution of strokes in the network, which is a fast running operation and can eliminate the checkerboard effect. We show the network architecture of the neural renderer in Figure 6 (c).

### 4.2. Stroke Design

Strokes can be designed as a variety of curves or geometries. In general, the parameter of a stroke should include the position, shape, color, and transparency.

We design a stroke represent of quadratic Bézier curve (QBC) with thickness to simulate the effects of brushes. The shape of the Bézier curve is specified by the coordinates of control points. Formally, the stroke is defined as

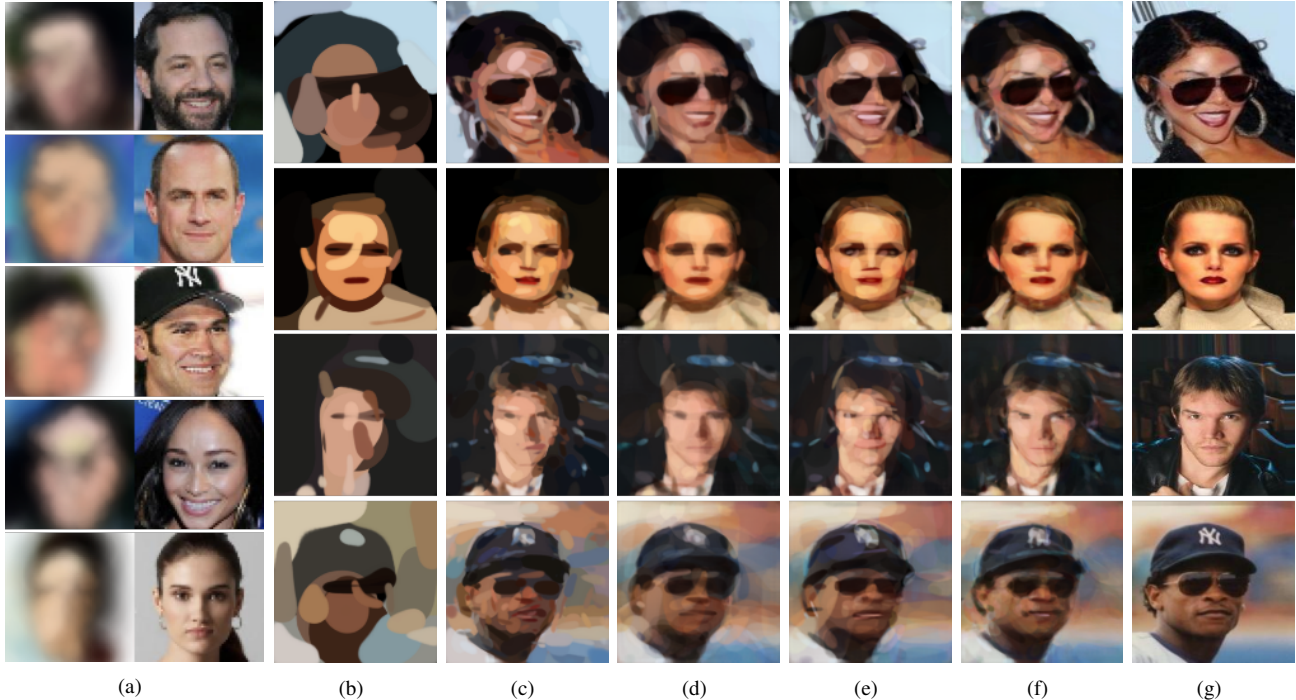


Figure 7: CelebA paintings under different settings. (a) The paintings of SPIRAL with 20 strokes [4] (b) Ours with 20 opaque strokes (c) Ours with 200 opaque strokes (d) Ours with 200 strokes and  $\ell_2$  reward (e) Ours with 200 strokes (Baseline) (f) Ours with 1000 strokes (g) The target images

the following tuple:

$$a_t = (x_0, y_0, x_1, y_1, x_2, y_2, r_0, t_0, r_1, t_1, R, G, B)_t, \quad (5)$$

where  $(x_0, y_0, x_1, y_1, x_2, y_2)$  are the coordinates of the three control points of the QBC.  $(r_0, t_0), (r_1, t_1)$  control the thickness and transparency of the two endpoints of the curve, respectively.  $(R, G, B)$  controls the color. The formula of QBC is:

$$B(t) = (1-t)^2 P_0 + 2(1-t)t P_1 + t^2 P_2, 0 \leq t \leq 1, \quad (6)$$

As changing stroke representation only requires changing the final stroke rendering layer, we can use neural renders with the same network structure to implement the rendering of different stroke designs.

## 5. Experiments

Four datasets are used for our experiments, including MNIST [16], SVHN [24], CelebA [23] and ImageNet [25]. We show that the agent has excellent performance in painting various types of real-world images.

### 5.1. Datasets

MNIST contains 70,000 examples of hand-written digits, where 60,000 are training data, and 10,000 are testing data. Each example is a grayscale image of  $28 \times 28$  pixels.

SVHN is a real-world streetview house number image dataset, including 600,000 digits images. Each sample in the Cropped Digits set is a color image of  $32 \times 32$  pixels. We randomly sample 200,000 images for our experiments.

CelebA contains approximately 200,000 celebrity face images. The officially provided center-cropped images are used in our experiments.

ImageNet (ILSVRC2012) contains 1.2 million natural scene images, which fall into 1000 categories. The extreme diversity of ImageNet poses a grand challenge to the painting agent. We randomly sample 200,000 images that cover 1,000 categories as training data.

In our task, we aim to train an agent that can paint any images rather than only the ones in the training set. Thus, we additionally split out testing set to test the generalization ability of the trained agent. For MNIST, we use the officially defined testing set. For other datasets, we randomly split out 2,000 images as the testing set.

### 5.2. Training

We resized all images to the resolution of  $128 \times 128$  pixels before feeding the agent. With an action bundle containing 5 strokes, it takes about 2.1s to paint an image using 200 strokes on a 2.2GHz Intel Core i7 CPU. On an NVIDIA 2080Ti GPU, a  $9.5\times$  acceleration can be achieved. The computation cost of the actor and renderer are about 554

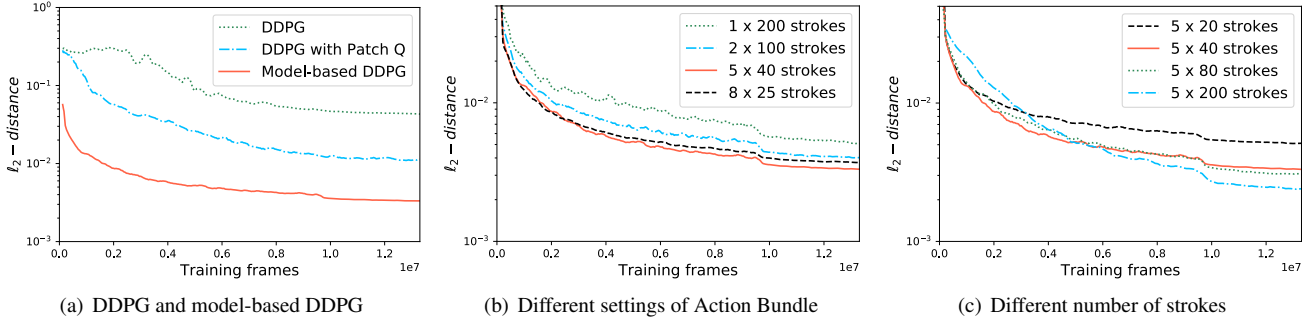


Figure 8: The testing  $\ell_2$ -distance between paintings and the target images of CelebA for ablation studies.

MFLOPs and 217 MFLOPs respectively for painting an action bundle.

We trained the agent with  $2 \times 10^5$  mini-batches for ImageNet and CelebA datasets,  $10^5$  mini-batches for SVHN and  $2 \times 10^4$  mini-batches for MNIST. Adam [15] was used for optimization, and the minibatch size was set as 96. The agent training was performed on a single GPU. It took about 40 hours for training on ImageNet and CelebA, 20 hours for SVHN and two hours for MNIST. It took 5 to 15 hours to train the neural renderer for every stroke design. The same trained renderer can be used for different agents.

At each iteration, we update the critic, actor, and discriminator in turn. All models are trained from scratch. The replay memory buffer was set to store the data of the latest 800 episodes for training the agent. Please refer to the supplemental materials for more training details.

### 5.3. Results

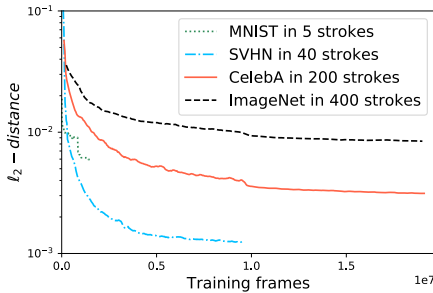


Figure 9: The testing  $\ell_2$ -distance between the paintings and the target images for different datasets.

The images of MNIST and SVHN show simple image structures and regular contents. We train one agent that paints five strokes for images of MNIST, and another one that paints 40 strokes for images of SVHN. The example paintings are shown in Figure 3 (a) and (b). The agents can perfectly reproduce the target images.

In contrast, the images of CelebA have more complex structures and diverse contents. We train a 200-strokes

agent to deal with the images of CelebA. As shown in Figure 3 (c), the paintings are quite similar to the target images although losing a certain level of details.

We train a 400-strokes agent to deal with the images of ImageNet, due to the extremely complex structures and diverse contents. As shown in Figure 3 (d), paintings are similar to the target images concerning the outline and colors of objects and backgrounds. Despite the loss of some textures, the agent still shows great power in decomposing complicated scenes into strokes and can reasonably repaint them.

We show the test loss curves of agents trained on different datasets in Figure 9.

In [4] SPIRAL shows its performance on CelebA. To make a fair comparison, we also train a 20-strokes agent as SPIRAL and use opaque strokes. The results of the two methods are shown in Figure 7 (a) and (b) respectively. Our  $\ell_2$  distance is 3x smaller than that of SPIRAL. We analyze the main differences between SPIRAL and our methods as follows. First, SPIRAL uses an undifferentiable painting simulator and have to use model-free RL algorithms, which usually perform worse than the model-based ones. Second, SPIRAL predicts an action by recurrently producing each dimension with strong nonlinearities. Our method simplifies this step by predicting multiple action vectors in just one step. Third, we believe having a sufficient number of strokes is critical for vivid results. SPIRAL only gets a reward after finishing a whole episode. This makes the rewards too sparse as the number of steps increases.

### 5.4. Ablation Studies

In this section, we study how the components or tricks affect the performance of the agent. The control experiments are performed on CelebA.

#### 5.4.1 Model-based vs. Model-free DDPG

We explore how much benefits are brought by model-based DDPG over original DDPG. Original DDPG can only essentially model the environment with observations and rewards from the environment. Besides, the high-dimensional

action space also stops model-free methods from successfully dealing with the painting task. To further explore the capability of model-free methods, we improve original DDPG with a method inspired by PatchGAN. We split the images into patches before feeding the critic, then use the patch-level rewards to optimize the critic. We term this method as PatchQ. PatchQ boosts the sample efficiency and improves the performance of the agent by providing much more supervision signals in training.

We show the performance of agents trained with different algorithms in Figure 8 (a). Model-based DDPG achieves the best performance, with  $5\times$  smaller  $\ell_2$  distance than DDPG with PatchQ, and  $20\times$  smaller  $\ell_2$  distance than the original DDPG. Although underperforming the model-based DDPG, DDPG with PatchQ outperforms original DDPG with significant margins.

#### 5.4.2 Rewards

$\ell_2$  distance is an alternative to the reward for training the actor. We show the painting results of using WGAN rewards (Section 3.3.3) and  $\ell_2$  rewards in Figure 7 (d) and (e) respectively. The paintings with WGAN rewards show richer textures and look more vivid. Interestingly, we find using WGAN rewards to train the agent can achieve a lower  $\ell_2$  loss on the testing data than using  $\ell_2$  rewards directly. This shows that WGAN distance is a better metric in measuring the differences between paintings and real-world images than  $\ell_2$  distance.

#### 5.4.3 Stroke Number and Action Bundle

The stroke number for painting is critical for the final painting quality, especially for texture-rich images. We train agents that can paint 100, 200, 400 and 1000 strokes, and the testing loss curves are shown in Figure 8 (c). It's observed that larger stroke numbers contribute to better painting quality. We show the paintings with 200-strokes and 1000-strokes in Figure 8 (e) and (f) respectively. To the best of our knowledge, few methods can handle such a large number of strokes. More strokes help reconstruct the details in the paintings.

We show test loss curves of several settings of Action Bundle in Figure 8 (b). We find that making the agent predict five strokes in one bundle achieves the best performance. We conjecture that increasing strokes number in one bundle helps the agent to build long-term plans as there will be fewer rounds of decision, even though it will increase the difficulty in a single round of decision. Thus, to achieve a trade-off, a few strokes in one bundle is a good setting for the agent. Experiments determine that setting five actions in an Action Bundle is optimal in our setting for the painting task.

#### 5.4.4 Alternative Stroke Representations

Besides the QBC, we find alternative stroke representations can also be well mastered by the agent, including straight strokes, circles, and triangles. We train one neural renderer for each stroke representation. The paintings with these renderers are shown in Figure 10. The QBC strokes produce excellent visual effects. Meanwhile, other stroke designs create different artistic effects. Although with different styles, the paintings still resemble the target images. This shows that our network architecture generalizes to other choices of stroke designs.

Also, by restricting the transparency of strokes, we can get paintings with different stroke effects, such as ink painting and oil painting as shown in Figure 7 (c).

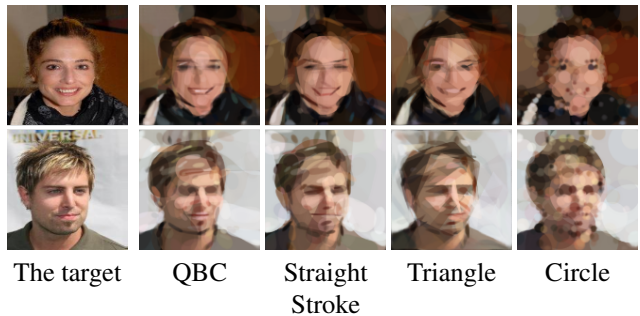


Figure 10: CelebA paintings using different stroke designs.

## 6. Conclusion

In this paper, we train agents that decompose the target image into an ordered sequence of strokes in a fashion mimicking human painting processes on canvases. The training is based on the Deep Reinforcement Learning framework, which encourages the agent to make long-term plans for sequential stroke-based painting. In addition, we build a differentiable neural renderer to render the strokes, which allows using model-based DRL algorithms to further improve the quality of recreated images. The learned agent can predict hundreds or even thousands of strokes to generate a vivid painting. Experimental results show that our model can handle multiple types of target images and achieve good performance in painting real-world images like human portraits and texture-rich natural scenes.

## References

- [1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 214–223. JMLR. org, 2017. 4
- [2] Alex Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. 4



- [3] Yajing Chen, Shikui Tu, Yuqi Yi, and Lei Xu. Sketchpix2seq: a model to generate sketches of multiple categories. *arXiv preprint arXiv:1709.04121*, 2017. 1
- [4] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. In *International Conference on Machine Learning*, pages 1652–1661, 2018. 1, 2, 6, 7
- [5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 1
- [6] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017. 5
- [7] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017. 1, 2
- [8] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc., 2018. <https://worldmodels.github.io>. 4
- [9] Paul Haeberli. Paint by numbers: Abstract image representations. In *ACM SIGGRAPH computer graphics*, volume 24, pages 207–214. ACM, 1990. 2
- [10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5
- [11] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460. ACM, 1998. 2
- [12] Aaron Hertzmann. A survey of stroke-based rendering. *IEEE Computer Graphics and Applications*, 23(4):70–81, July 2003. 2
- [13] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 448–456. JMLR. org, 2015. 5
- [14] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 5
- [15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7
- [16] Yann LeCun. The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>, 1998. 3, 6
- [17] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, and Zehan Wang. Photo-realistic single image super-resolution using a generative adversarial network. 2016. 1
- [18] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017. 4
- [19] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 2
- [20] Liang Lin, Kun Zeng, Han Lv, Yizhou Wang, Yingqing Xu, and Song-Chun Zhu. Painterly animation using video semantics and feature correspondence. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, pages 73–80. ACM, 2010. 2
- [21] Peter Litwinowicz. Processing images and video for an impressionist effect. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 407–414. ACM Press/Addison-Wesley Publishing Co., 1997. 2
- [22] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Advances in Neural Information Processing Systems*, pages 9628–9639, 2018. 5
- [23] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 3, 6
- [24] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bis-sacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011. 3, 6
- [25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 3, 6
- [26] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016. 5
- [27] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016. 5
- [28] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014. 4

- [29] Jifei Song, Kaiyue Pang, Yi-Zhe Song, Tao Xiang, and Timothy M Hospedales. Learning to sketch with shortcut cycle consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 801–810, 2018. [1](#)
- [30] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992. [4](#)
- [31] Sitao Xiang and Hao Li. On the effects of batch and weight normalization in generative adversarial networks. *arXiv preprint arXiv:1704.03971*, 2017. [5](#)
- [32] Ning Xie, Hirotaka Hachiya, and Masashi Sugiyama. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. *IEICE TRANSACTIONS on Information and Systems*, 96(5):1134–1144, 2013. [2](#)
- [33] Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5485–5493, 2017. [4](#)
- [34] Ningyuan Zheng, Yifan Jiang, and Dingjiang Huang. Strokenet: A neural painting environment. In *International Conference on Learning Representations*, 2019. [1](#), [2](#)
- [35] Tao Zhou, Chen Fang, Zhaowen Wang, Jimei Yang, Byungmoon Kim, Zhili Chen, Jonathan Brandt, and Demetri Terzopoulos. Learning to doodle with deep q-networks and demonstrated strokes. [1](#), [2](#)

## 7. Appendix

### 7.1. Architecture

The network structure diagrams are shown in Figure 11, 12, 13 and 14, where FC refers to a fully-connected layer, Conv is a convolution layer. The hyperparameters used in training are listed as much as possible in Table 1 and 2. All ReLU activations between the layers have been omitted for brevity.

Table 1: Hyper-parameters for the **DDPG**.

# Action per step	5
# Step per episode	40
Replay buffer size	800 episodes
# Training batches	$2e5$
Batch size	96
Actor learning rate	$\{3e-4, 1e-4\}$
Critic learning rate	$\{1e-3, 3e-4\}$
	* Learning rate decays after $1e5$ training batches
Reward discount factor	$0.95^5$
Optimizer	Adam
Actor Normalization	BN
Critic Normalization	WN with TReLU

Table 2: Hyper-parameters of the **discriminator** training.

Replay buffer size	800 episodes
# Training batches	$2e5$
Batch size	96
Learning rate	$1e-4$
Optimizer	Adam ( $\beta_1 = 0.5, \beta_2 = 0.999$ )
Normalization	WN with TReLU

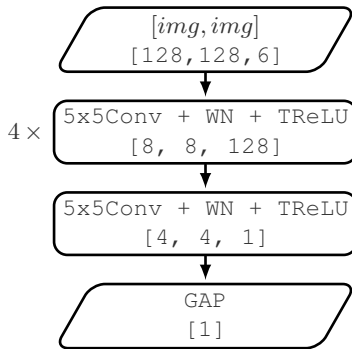


Figure 11: The network architecture of the **discriminator**. GAP refers to a global average pooling layer. Input contains two images.

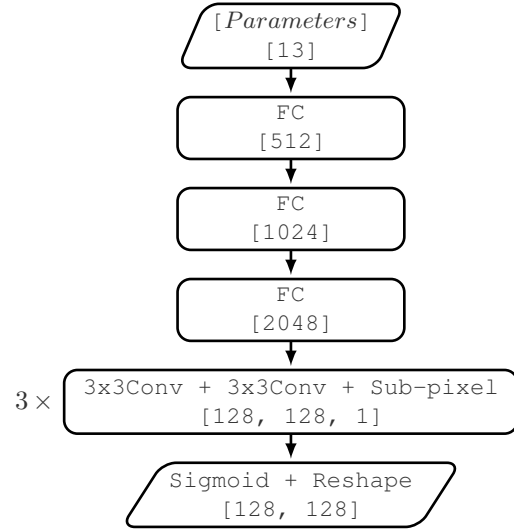


Figure 12: The network architecture of the **neural renderer**.

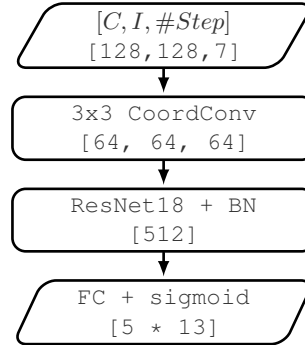


Figure 13: The network architecture of the **actor**. Input contains the canvas, target image and step number.

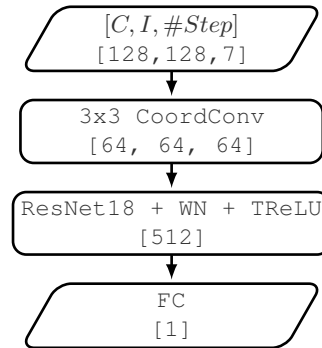


Figure 14: The network architecture of the **critic**. Input contains the canvas, target image and step number.